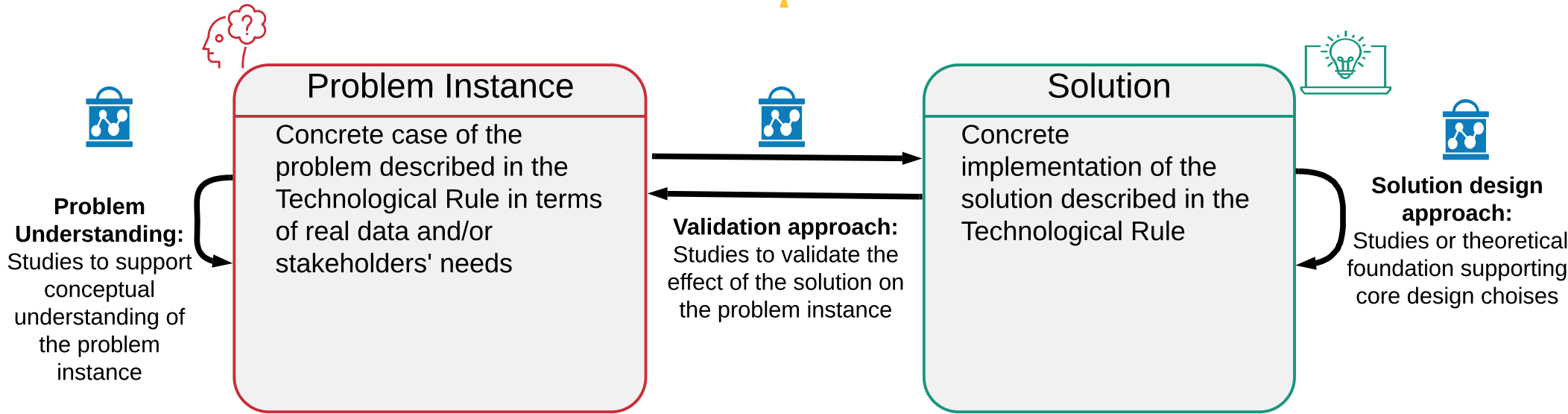


Generic Visual Abstract



Technological rule: To achieve effect/change in situation/context apply solution/intervention



Relevance: Characteristics of the context that are likely to impact applicability and potential value of the proposed solution



Rigor: Characteristics of the three knowledge creating activities (problem understanding, solution design and in context evaluation) that adds to the strength of the empirical support of the Technological Rule



Novelty: Positioning of the Technological Rule in terms of previous knowledge



Technological rule: To achieve effect/change in situation/context use solution/intervention



Problem Understanding:

Problem Instance



Validation approach:

Solution



Solution design approach:



Relevance:



Rigor:

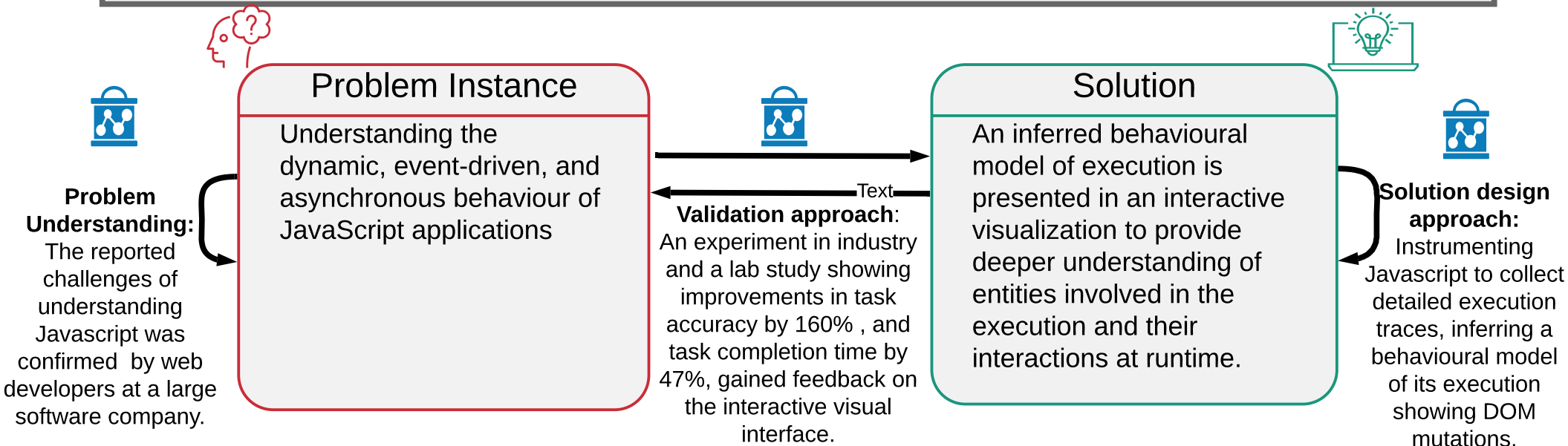


Novelty:

Paper Title:



Technological rule: To improve developers comprehension when understanding event-based interactions in JavaScript capture and visualize low-level event-based interactions mapped to a higher level behavioural model.



Relevance: Common [JavaScript] comprehension tasks are supported by the proposed visualization approach, which could help many web developers.



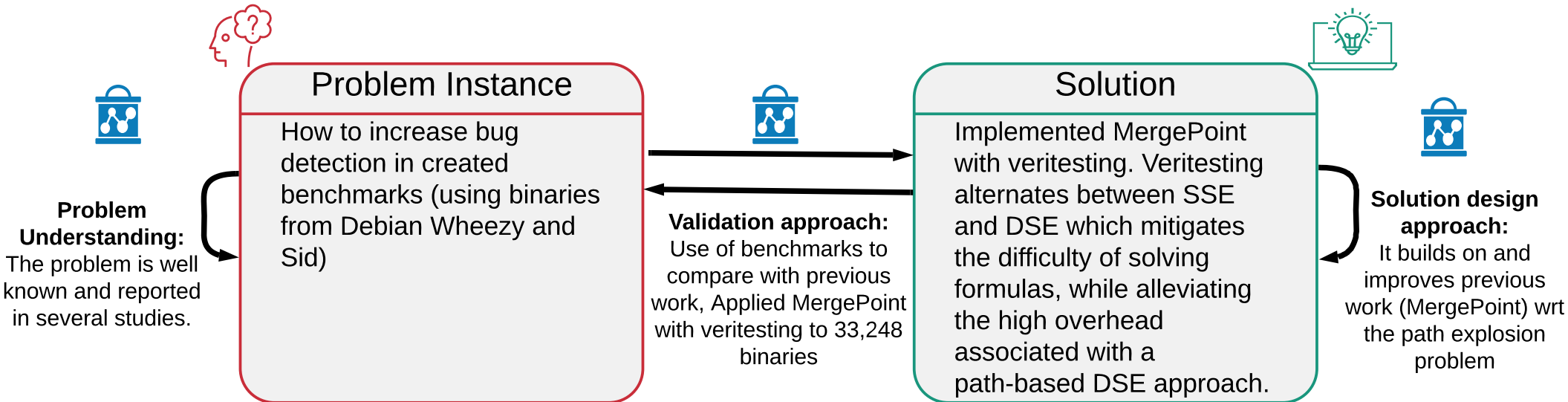
Rigor: Two controlled experiments were conducted with 34 users to measure and compare task completion time and accuracy, with preferred tools selected by a control group.



Novelty: 1) Capturing and inferring temporal and causal relations between JavaScript execution, DOM mutations, and server communications; and 2) A behavioural model of program behaviour, displayed as an interactive visualization in an open source tool.



Technological rule: To increase efficiency and effectiveness when using symbolic execution alternate between static and dynamic symbolic execution



Relevance: Lack of efficiency when using symbolic execution



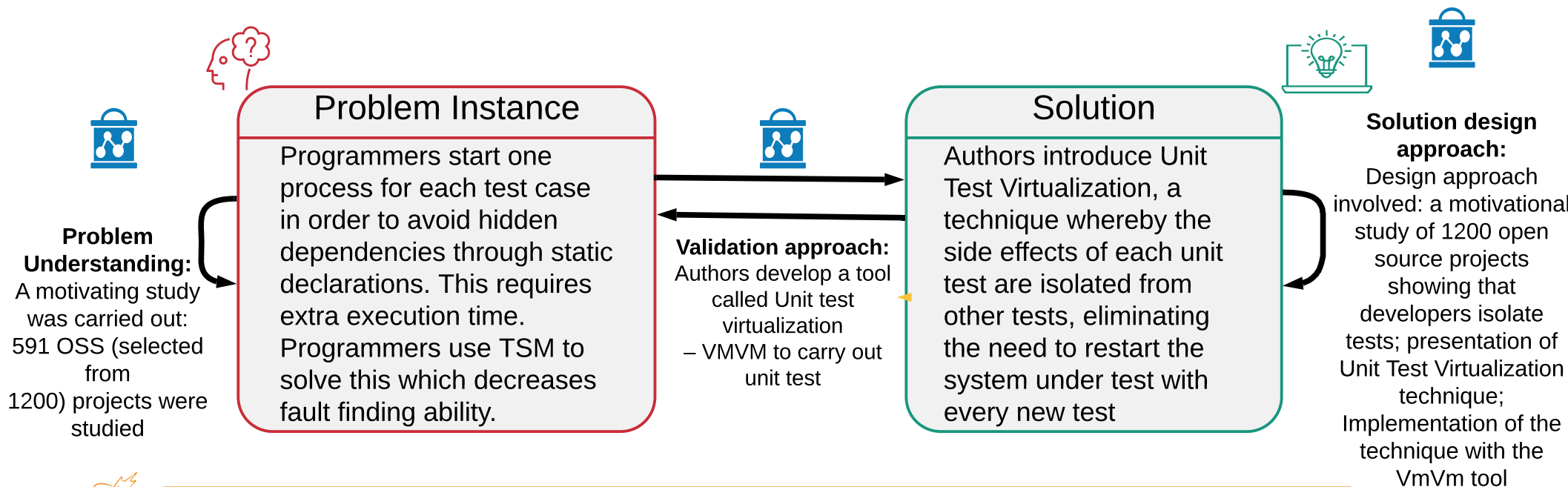
Rigor: Large-scale experiment on 33,248 programs from Debian Linux. MergePoint generated billions of SMT queries, hundreds of millions of test cases, millions of crashes, and found 11,687 distinct bugs



Novelty: Tested an order of magnitude more applications than have been tested by prior symbolic execution research; analyzed each application for less than 15 minutes per experiment; and improved open source software by finding over 10,000 bugs and generating millions of test cases.



Technological rule: To reduce overhead during unit testing use unit test virtualization



Relevance: Test case selection and isolation for developers working on large applications with long running test suites



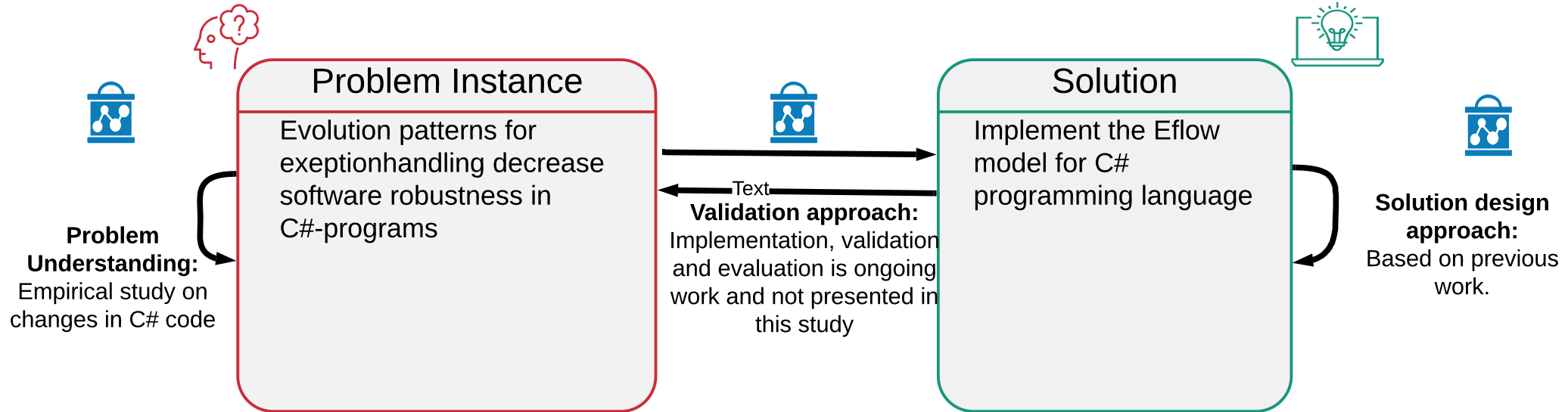
Rigor: Two Empirical studies to evaluate performance of VMVM vs. TSM wrt: reduction of execution time (RT); reduction in fault-finding ability (RF); reduction in test suite size (TS). One study conducted on 19 versions of 4 applications; a second study conducted on 20 OS Java applications.



Novelty: 1) Motivational study of the test suites of 1,200 open source projects showing that developers isolate tests 2.) Unit Test Virtualization technique to efficiently isolate test cases 3.) Implementation and evaluation of VmVm tool to show its efficacy in reducing test suite runtime and maintaining fault-finding properties



Technological rule: To improve the simultaneous satisfaction of software maintainability and reliability in built-in EHM in programming languages apply explicit exception channels, which support modular representation of global exceptional-behaviour properties.



Relevance: Built-in exception handling mechanisms affect robustness negatively. Relevant for mainstream programming languages (C#, Ruby, Python and many others) that provide built-in exception handling mechanisms



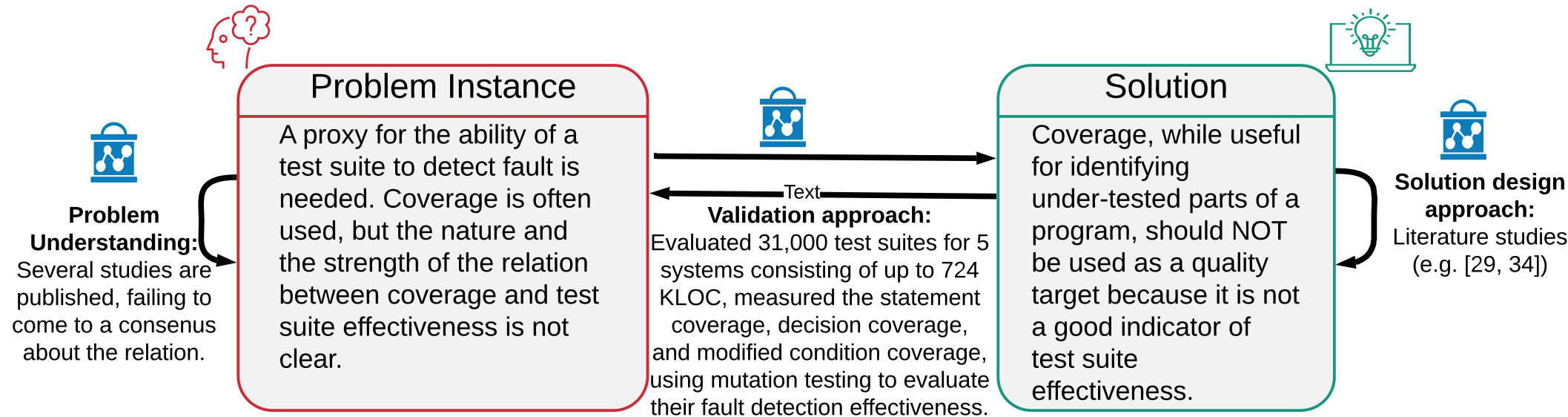
Rigor: Changes to both normal and exceptional code in 119 versions extracted from 16 software projects were analysed. Analysis was based on common models for exception handling mechanisms.



Novelty: Empirical knowledge on how programs using maintenance-driven facilities for exception handling evolve over time (impact and side effects on robustness)



Technological rule: When assessing test suite effectiveness in software testing of Java programs do NOT rely on coverage measures



Relevance: Using coverage for understanding of the effectiveness of automatic unit tests by practitioners planning unit tests



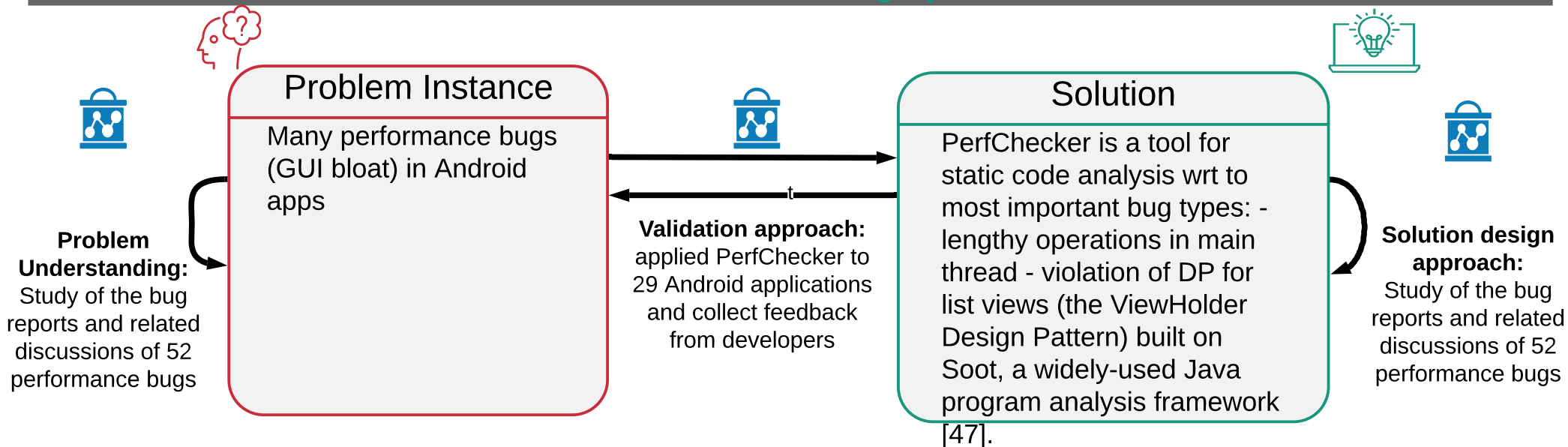
Rigor: The recommendation is ensured by investigating a large set of (generated) mutants of 5 programs, and test cases for three programs.



Novelty: The negative rule has previously been seen as a commonly accepted fact.



Technological rule: To achieve improved performance bug identification ability in Android app development apply static analysis wrt lengthy oper. in the main thread and violations of the View-holder design pattern



Relevance: A previous inspection of 60,000 Android applications [11] revealed that 11,108 of them have suffered or are suffering from performance bugs. Many smartphone applications are developed by small teams without dedicated quality assurance. These developers lack viable techniques to help analyze the performance of their applications [23].



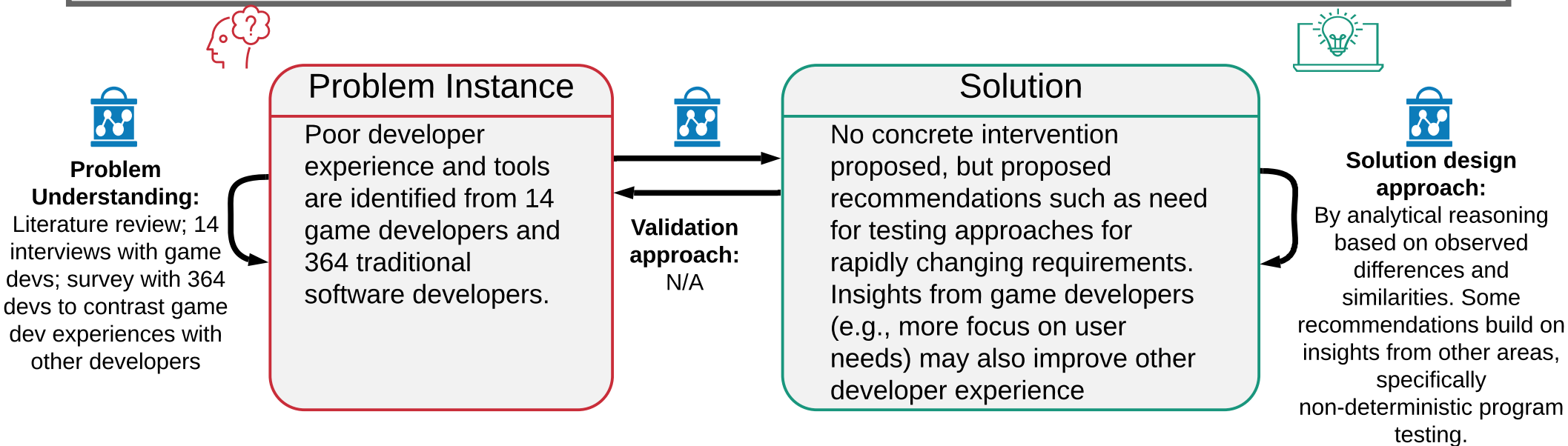
Rigor: PerfChecker was applied to analyze the latest version of all 29 Android applications, which comprise more than 1.1 million lines of Java code. Selection of OSS projects based on relevant factors, conclusions based on tests of significance.



Novelty: PerfChecker finished analyzing each application within a few minutes and detected 126 matching instances of the two performance bug patterns in 18 of the 29 applications.



Technological rule: To improve developer experience and tools during game and other development scenarios apply the recommendations that emerged from this study.



Relevance: Software developers in general wishing to improve their processes and tools.



Rigor: Triangulation of methods: surveyed a large and diverse group of game developers to validate insights from the literature and interviews. Survey and interview questions are posted online to support future replication.



Novelty: New insights on game developer challenges and practices.



Technological rule: To improve the responsiveness of asynchronous programs, where outdated or misused idioms are adopted, use automated tools to detect and fix these issues.



Problem Understanding: 1378 open source apps (Microsoft CodePlex and GitHub) were analyzed, finding many anti-patterns of misused asynchronous programming.

Problem Instance

Many instances of misuse of asynchronous programming antipatterns are identified and studied in a corpus of open source apps.



Validation approach: Evaluation of the effectiveness and efficiency of tools with the code corpus used to identify the problems. Patches are suggested to the original developers (via a pull request), many of which were accepted.

Solution

Two tools are proposed: 1) a refactoring tool called Asyncifier and 2) a transformation tool called Corrector to fix misuse antipatterns



Solution design approach: A large scale study of WPApps analyzing how older idioms and misuses occur was conducted. Results led to a toolkit to address those issues.



Relevance: Asynchronous programs that use outdated asynchronous idioms, or misuse modern idioms. The techniques they propose can be applied to other C# platforms. Relevant for developers as well as tool vendors, language and library designers, educators and researchers.



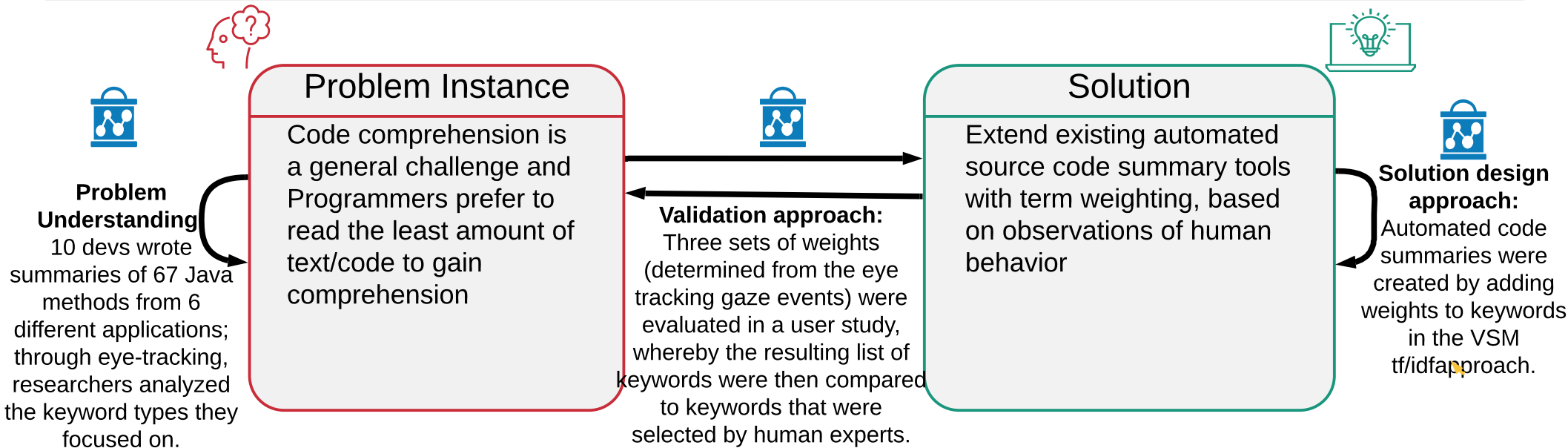
Rigor: An empirical study that quantitatively assesses that the techniques work, and a study to demonstrate applicability of their solution were carried out.



Novelty: 1) large scale empirical study to answer questions about asynchronous programming and async/wait; 2) development and evaluation of a toolkit (two tools) that implements analysis and transformation algorithms to address challenges



Technological rule: To improve automated source code summarization for programmers reading code apply weighting scheme for terms, based on human behavior



Relevance: Summaries of code are useful for programmers to aid comprehension



Rigor: Human experts read Java methods and ranked the top five most-relevant keywords from each method and compared to the automated summarization



Novelty: Weighting factors in automated code summarization, using VSM, based on empirical study of human behavior



Technological rule: To find factors to take into account by software quality assurance in large scale software projects take specific management actions to mitigate identified organizational problems that may impact software quality



Problem Understanding:

10-month study based on non-participant observation of a software development team's weekly status meetings, validated with manager interviews.

Problem Instance

To understand how organizational factors negatively impact software quality, a study was conducted at a large telecommunications company with over 40 years in business where it was observed that despite good developers, the outcome on software quality was not always positive.



Validation approach:

N/A. They do not evaluate their recommendations.

Solution

To address the observed flaws they suggest the following corrective actions: a) encourage face-to-face meetings, b) don't manage multiple versions of components, c+d) follow standardized processes for acquisition of third-party components, e) change processes gradually, f+g) manage scope, quality and deadlines, h) balance formal processes and human interaction, i,j) promote knowledge sharing.



Solution design approach:

They rely on their experience, insights gained during the study and the literature to arrive at recommendations to mitigate the problems they observed



Relevance: Large-scale, general software quality problems



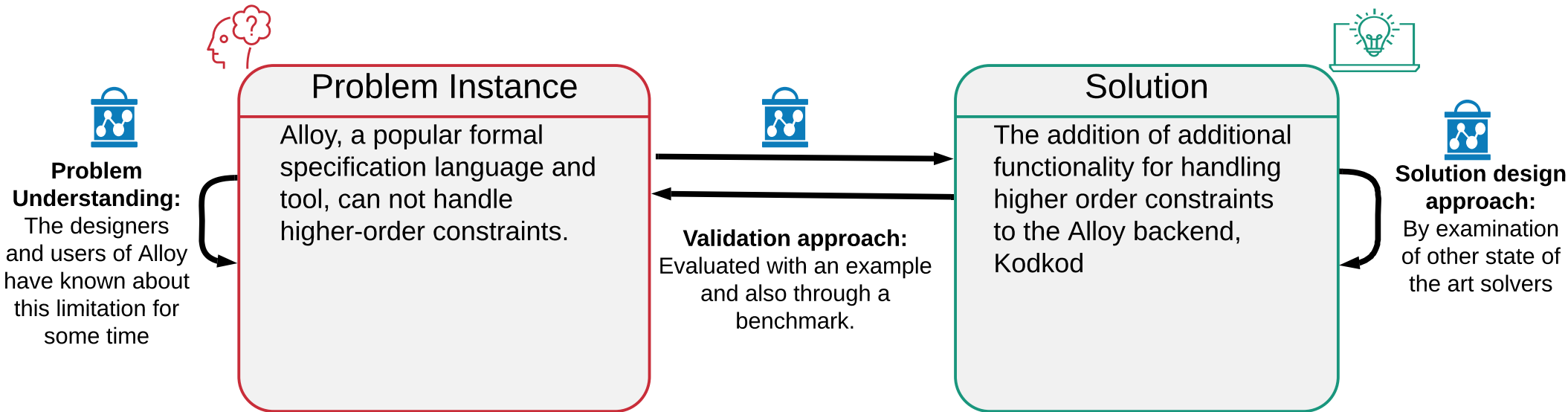
Rigor: Ten months of observation of an in-house software development project within a large telecommunications company to understand the problems.



Novelty: The problem description, based on a year of observations that shows organizational factors may impact software quality



Technological rule: To handle higher-order quantifiers in a generic and model-agnostic way, when using Alloy use the updated version that supports higher order constraints



Relevance: It is relevant to designers and users of Alloy who needs to handle higher order constraints..



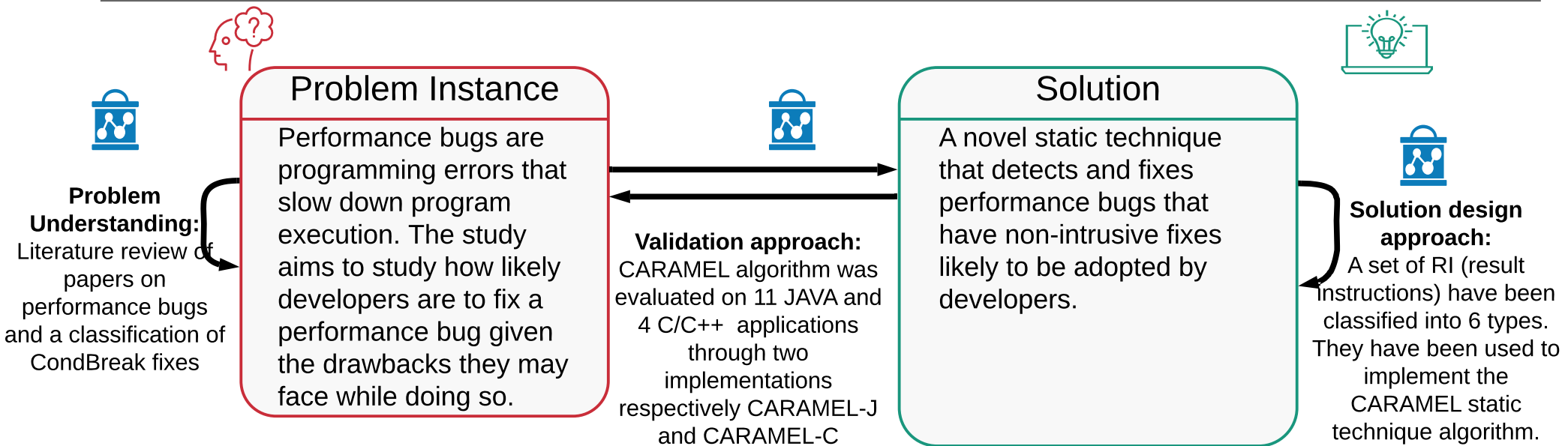
Rigor: The paper validates the approach on a community accepted benchmark of reasonably complex problems. The benchmark process considers three competing tools and one state of the art synthesis tool



Novelty: Alloy* is the first general-purpose constraint solver capable of solving formulas with higher order quantification. Existing solvers either do not admit such quantifiers, or fail to produce a solution in most cases.



Technological rule: To fix performance bug problems caused by unnecessary loop execution use a novel static technique that adds a break within loops (CondBreak fix)



Relevance: Programming errors due to performance bugs. Relevant for programmers for which performance is an issue.



Rigor: The CAMEL static technique algorithm was applied to 11 JAVA applications and 4 C/C++ applications and identified 61 performance bugs in the JAVA applications and 89 in C/C++ ones. Of these, 10+24 were not found before.



Novelty: (i) detect performance bugs whose fixes clearly offer more benefits than drawbacks to developers; (ii) identify a family of performance bugs; (iii) CAMEL algorithm as novel static technique for detecting performance bugs that have CondBreak fixes;



Technological rule: N/A N/A N/A



Problem Understanding:

Conducting a literature review and a survey among 79 program-committee and editorial-board members of 11 major software-engineering venues

Problem Instance

The SE community is lacking consensus on internal and external validity



Solution

There is no proposed intervention



Solution design approach:
N/A

Validation approach:
N/A



Relevance: N/A



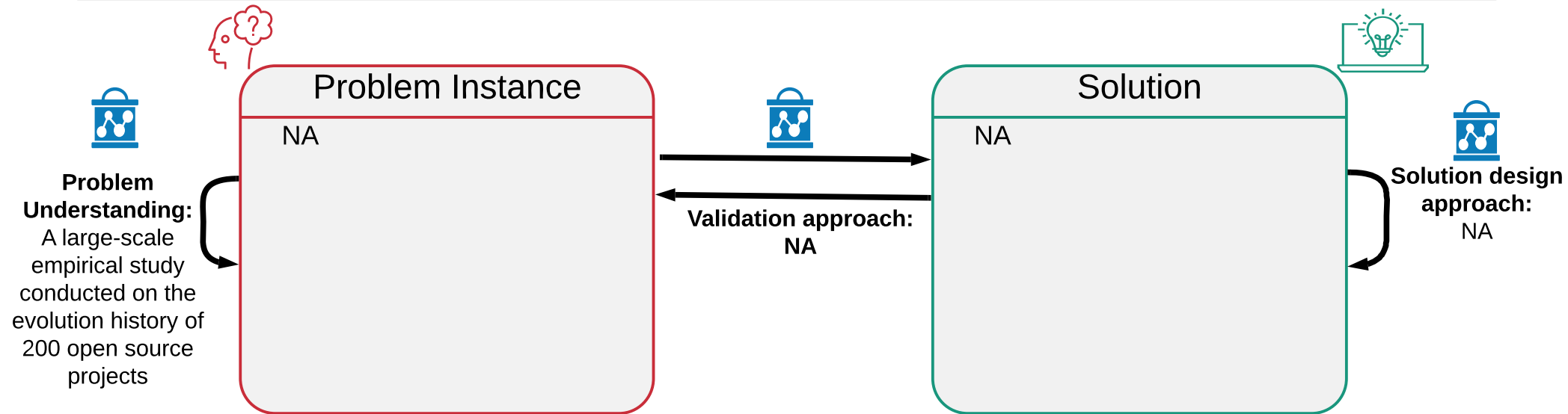
Rigor: SE experts were asked through an online survey.



Novelty: Description of the situation



Technological rule: To better plan activities for improving design and source code quality when developing software consider when and why smells are introduced



Relevance: The study context is the change history of 200 projects belonging to three software ecosystems, namely Android, Apache, and Eclipse



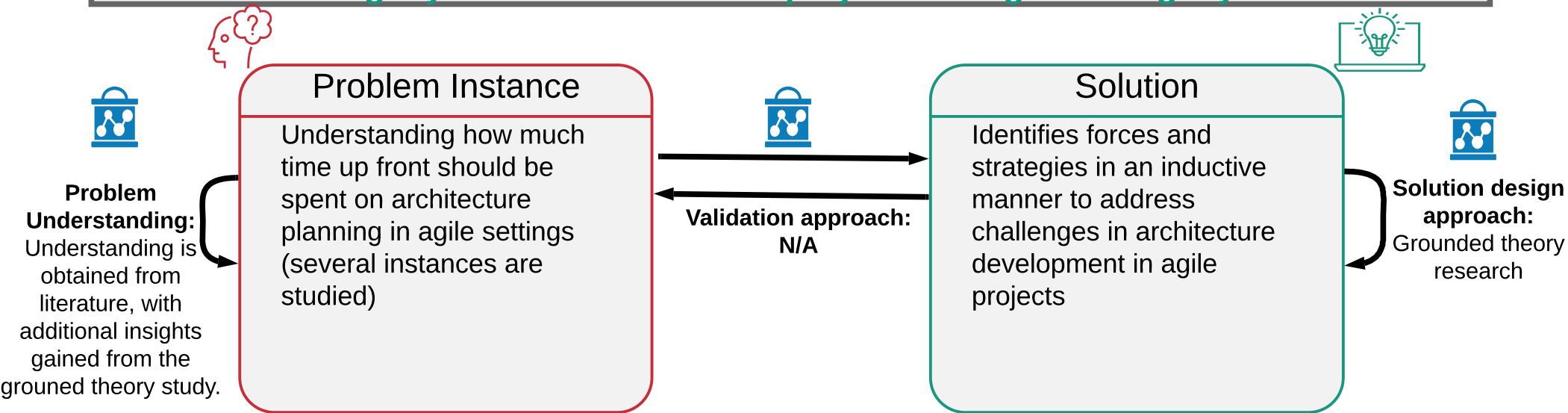
Rigor: Code smells and their evolution is analyzed in 200 projects that were extracted from three ecosystems: android, apache and eclipse.



Novelty: First comprehensive empirical investigation into when and why code smells are introduced in software projects.



Technological rule: To understand architecture development in agile development settings use a Theory that identified forces in the context of a project that affect agility and their relation to project strategies for agility



Relevance: Researchers and practitioners that are involved in architecture design



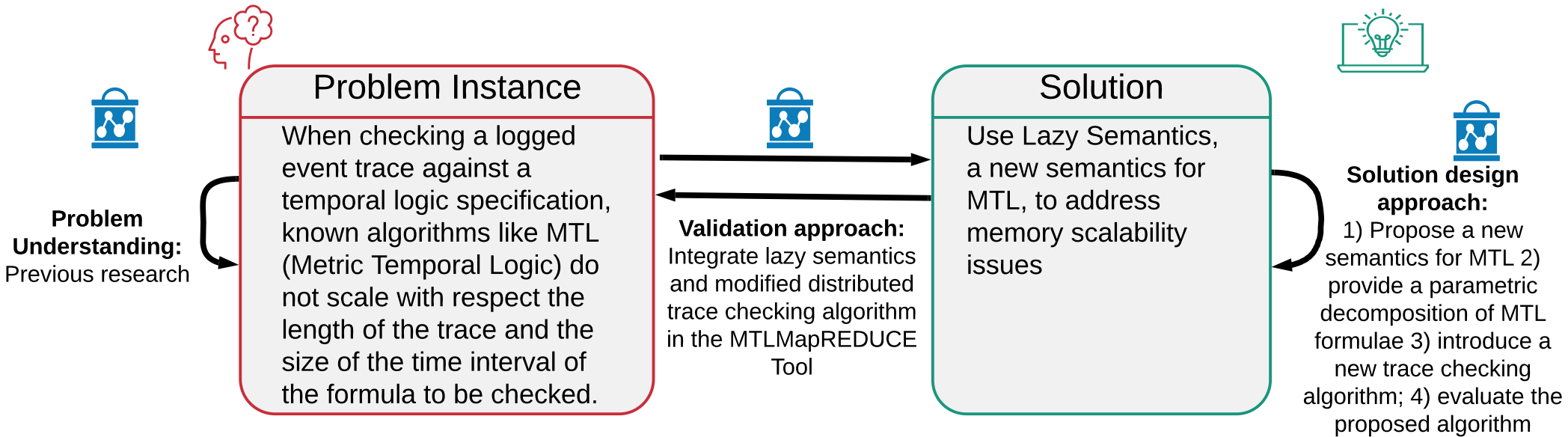
Rigor: Use of grounded theory with 44 participants to give insights on the problem instances.



Novelty: The presented theory.



Technological rule: In order to scale the size of traces when checking a logged event trace against a temporal logic specification use lazy semantics



Relevance: Problems related to tracking large scale log data for events. The technique is relevant to who inspects server logs, crash reports, and test traces in order to analyse problems at runtime.



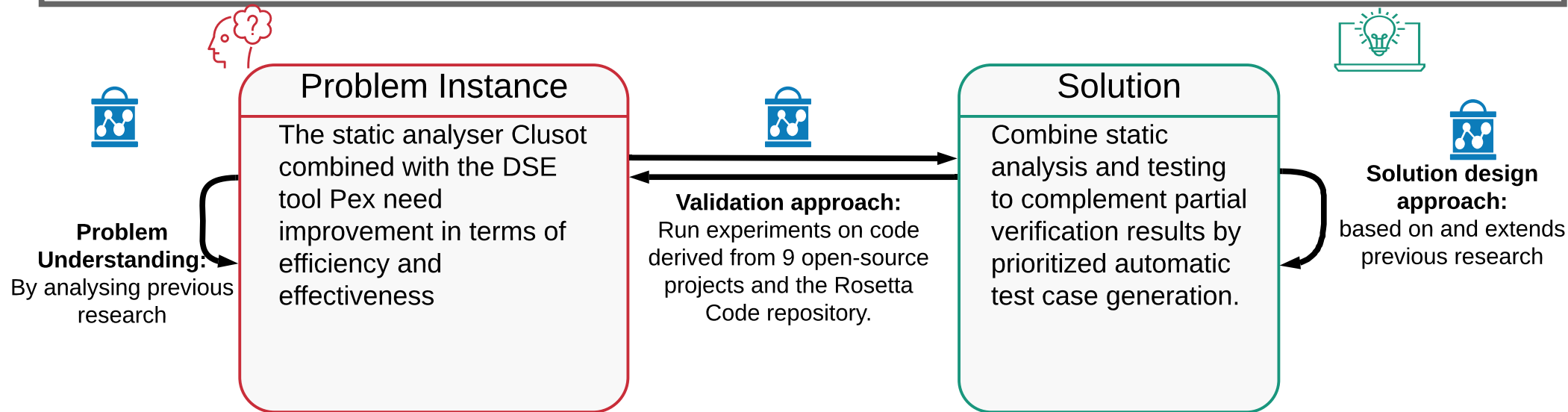
Rigor: Evaluation of properties, using 20 synthesized traces with 50 million elements each



Novelty: Use a new semantics for MTL called Lazy Semantics



Technological rule: To reduce testing time and improve coverage in programs that are partially verified based on unsound assumptions guide DSE toward unverified program executions



Relevance: Relevant for .net developers



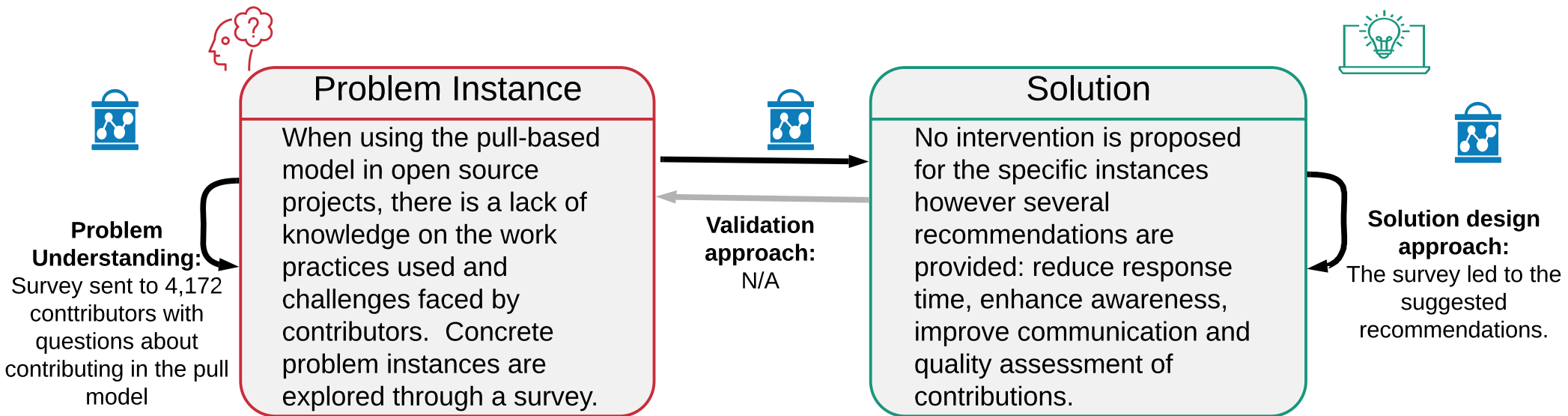
Rigor: Experiments used 101 methods written in C# from nine open-source projects and from solutions to 13 programming tasks on the Rosetta Code Repository



Novelty: In contrast to existing work, our technique supports the important case that the verification results are obtained by an unsound (manual or automatic) static analysis.



Technological rule: To improve the pull-based contribution model in open source projects, follow recommendations that may improve the process.



Relevance: Open source projects that accept contributions from the community wishing to improve their process.



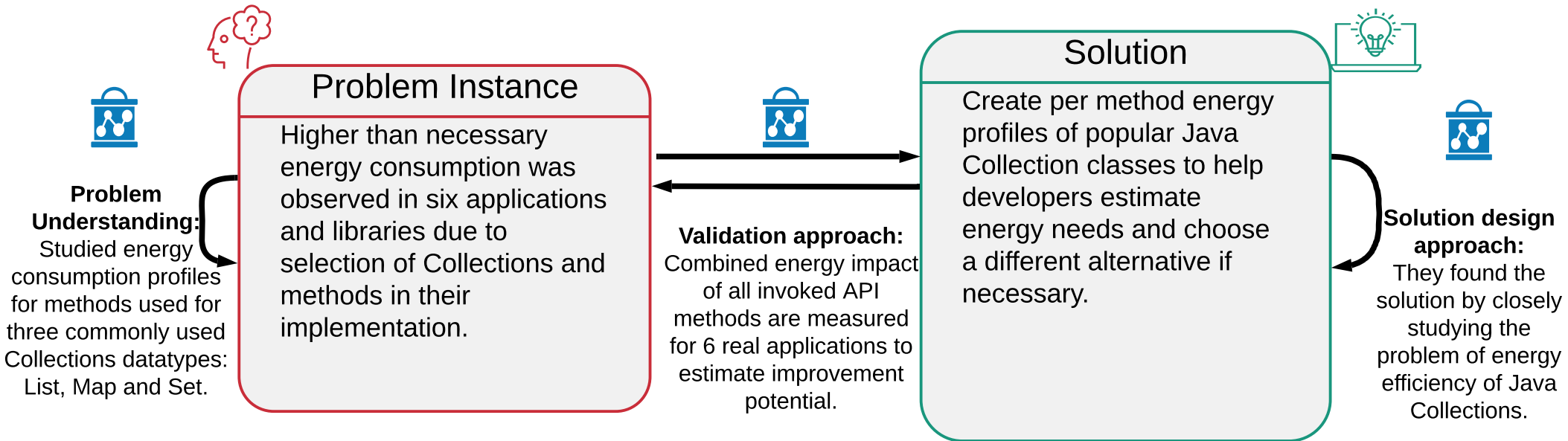
Rigor: Carefully designed survey, large sample of contributors, wide range of projects, pilot of survey



Novelty: Identified challenges in pull-based development and recommendations for improving the pull-based process.



Technological rule: To improve the energy efficiency of software developed in Java use per-method energy profiles



Relevance: Developers wishing to use green programming techniques (in particular to select among methods in Java Collections)



Rigor: They study if what they find applies in other cases by modifying those applications and checking if using the profiles actually helps improve the energy consumption various applications (e.g., Google Gson, XStream and K-9 Mail).



Novelty: A new approach for profiling Java Collections in terms of energy consumption.



Technological rule: To reduce instrumentation overhead for computing crash paths, for Javascript bugs that are difficult to diagnose from error messages and stack traces alone, compute crash paths by distributing the instrumentation overhead to a crowd of users, thereby reducing the impact of instrumentation at run time



Problem Understanding:
A case study

Problem Instance

Error messages and stack traces are insufficient to debug real-world specific problematic issues in JavaScript applications reported on Github



Solution

A prorotype implementation CROWDIE was used to debug the issues

Validation approach:
Evaluated on 10 real-world issues where diagnosis was problematic.
Measured instrumentation overhead and number of times a crash needs to be encountered to recover the complete crash path.



Solution design approach:
Inspired by similar crowdsourcing techniques for debugging that were pioneered by Liblit et al. and by Orso et al.



Relevance: Error messages and stack traces provide insufficient information for diagnosing problems in deployed Java applications



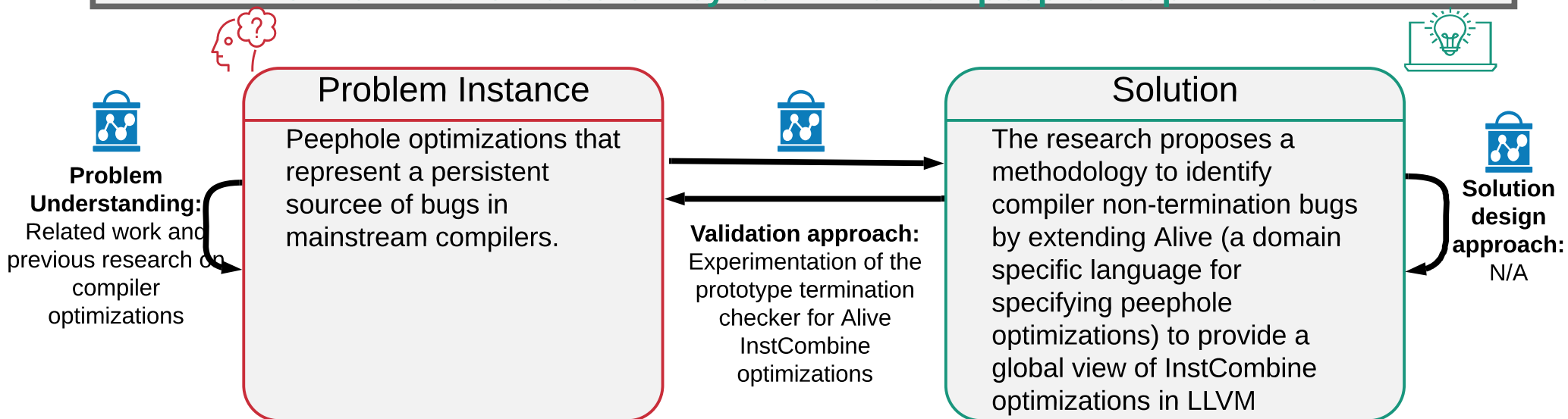
Rigor: Tool is used to debug 10 real world issues.



Novelty: The notion of crash paths is novel as well as application of crowd-based techniques to compute them.



Technological rule: In order to detect non-termination bugs with peephole optimizations use ALIVE tool that LLVM developers can use to check non-termination before they commit a new peephole optimization



Relevance: Relevant for relevant for LLVM developers when addressing non-termination bugs that arise when a suite of peephole optimizations is executed until a fixed point.



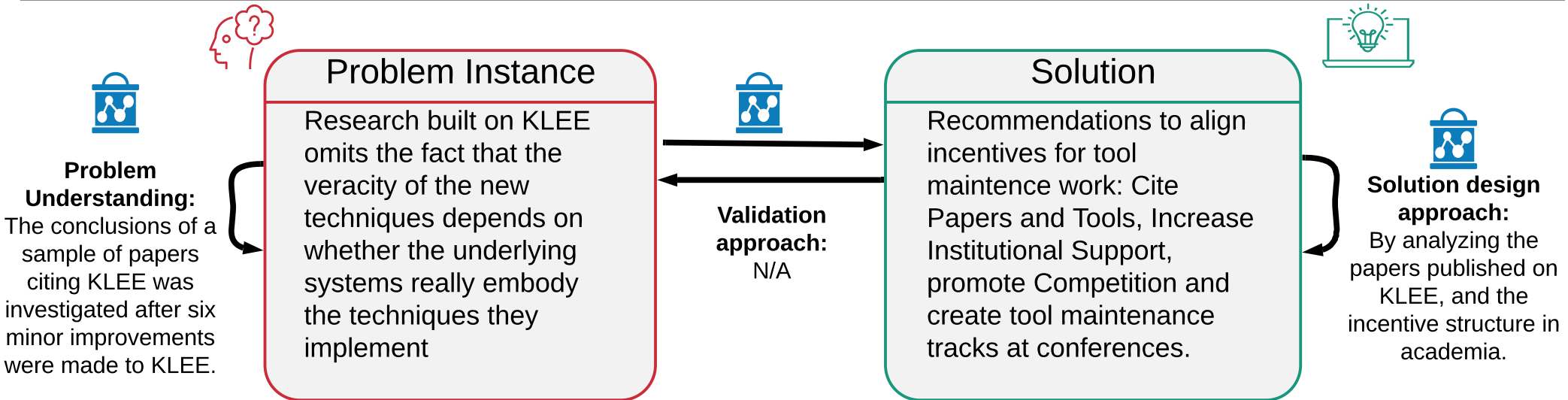
Rigor: 184 optimization sequences involving 38 optimizations that cause non-terminating compilation in LLVM with Alive-generated C++ code have been identified in the experimentation.



Novelty: 1) new class of compiler non-termination bugs resulting from lack of a global view of peephole optimizations in LLVM; 2) a methodology to detect compiler non-termination bugs building on top of Alive, which checks the correctness of each individual InstCombine transformation, 3) non-increasing source in the self-composition of a sequence of optimizations as the necessary condition for non-termination, 4) a technique to generate concrete inputs to demonstrate non-termination errors to aid debugging



Technological rule: To increase efficient replication and aggregation of knowledge in software engineering research, developing tools, change academic publication and rewarding structures



Relevance: Demonstrating the value of technique A may involve direct comparison with or building on technique B. The implementations of both A and B will then play a crucial role in the validity of the conclusions



Rigor: Previous studies based on KLEE were replicated after its improvement. Original authors were involved to give feedback.



Novelty: Empirical evidence that the strong emphasis on introducing "new" techniques may lead to wasted effort and questionable research conclusions.



Technological rule: To improve information retrieval in SE where there is a lexical gap between text documents and code use word embeddings based on the distributional hypothesis



Problem Instance

Bug localization tasks in an Eclipse corpus of code as well as linking questions to code snippets in StackOverflow.



Solution

Authors propose to use word embeddings based on the distributional hypothesis to improve API recommendations during tasks such as bug localization, and recommendations for APIs in StackOverflow.



Solution design approach:

Proposed solution is inspired by the distribution hypothesis (in NL processing) that co-occurrences of words (whether NL or code) may be related.

Validation approach:
Solutions has been evaluated on the Eclipse bug localization benchmark dataset.

Problem Understanding:
Literature proves that existing baseline methods do not effectively link text and code when there is a lexical gap in a bug localization benchmark.



Relevance: Relevant for tool designers, in situations where there is a lexical gap between search queries (usually expressed in textual form) and retrieved files (normally source code files).



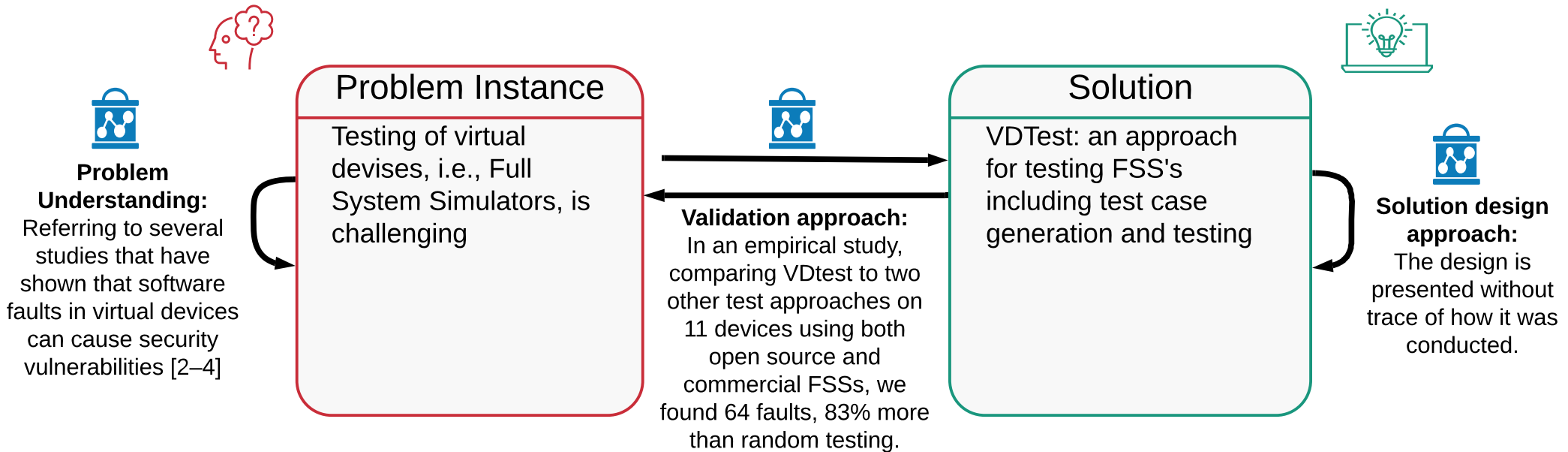
Rigor: train word embeddings on API documents, tutorials and reference documents, then aggregate in order to estimate semantic similarities between documents



Novelty: Word embeddings have been applied in various NLP tasks, but not for linking code files and textual queries in SE.



Technological rule: To improve test effectiveness in testing of virtual devices in full system simulators (FSS) use VCTest to obtain test cases and carry out testing



Relevance: Testing of virtual devices



Rigor: Empirical evaluation of VCTest to virtual devices from three FSSs that are widely used in both industry and academia



Novelty: The design of VCTest.



Technological rule: To understand how human brain processes software engineering tasks when making subjective human judgments use medical imaging techniques (fMRI - functional magnetic resonance imaging)



Problem Understanding: Controlled experiment with fMRI to gain understanding of research challenges.

Problem Instance

Subjective judgments in software engineering tasks are of critical importance but can be difficult to study with conventional means.



Validation approach:
N/A

Solution

Use medical imaging techniques to understand code comprehension and code review and the relationship to natural language and expertise.



Solution design approach:
N/A



Relevance: For researchers trying to understand how humans perform code review and program comprehension activities



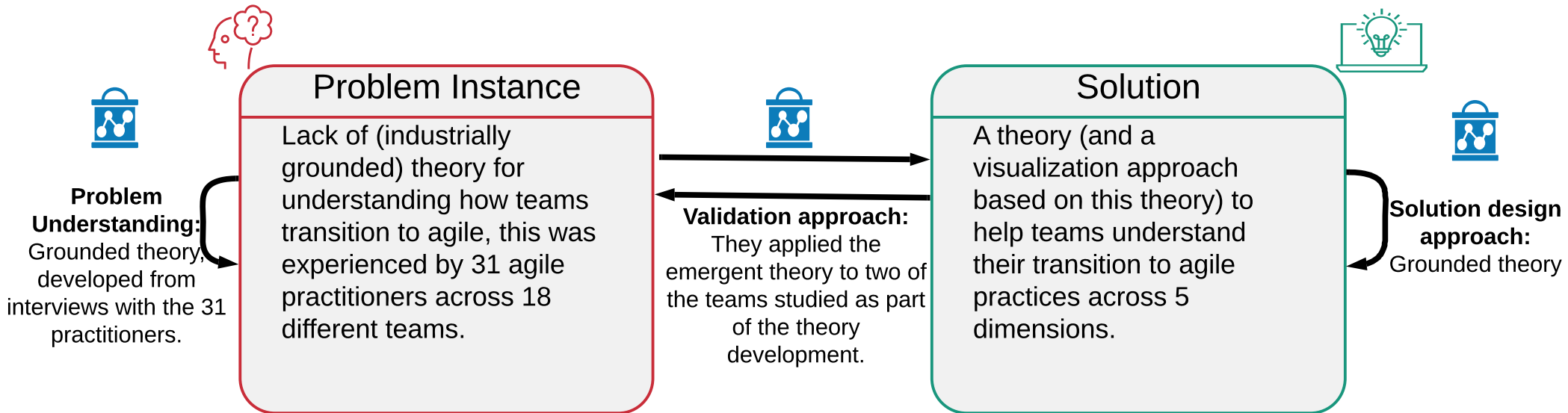
Rigor: A controlled experiment with 29 participants was conducted, where the authors examined code comprehension, code review and prose review activities using functional magnetic resonance imaging.



Novelty: Understanding how to measure brain activity for carrying out different tasks



Technological rule: To gain more understanding of a team's work practices, when transitioning to agile, use the 5 dimensions proposed in this theory.



Relevance: Industrial teams wishing to understand and improve their transition to agile



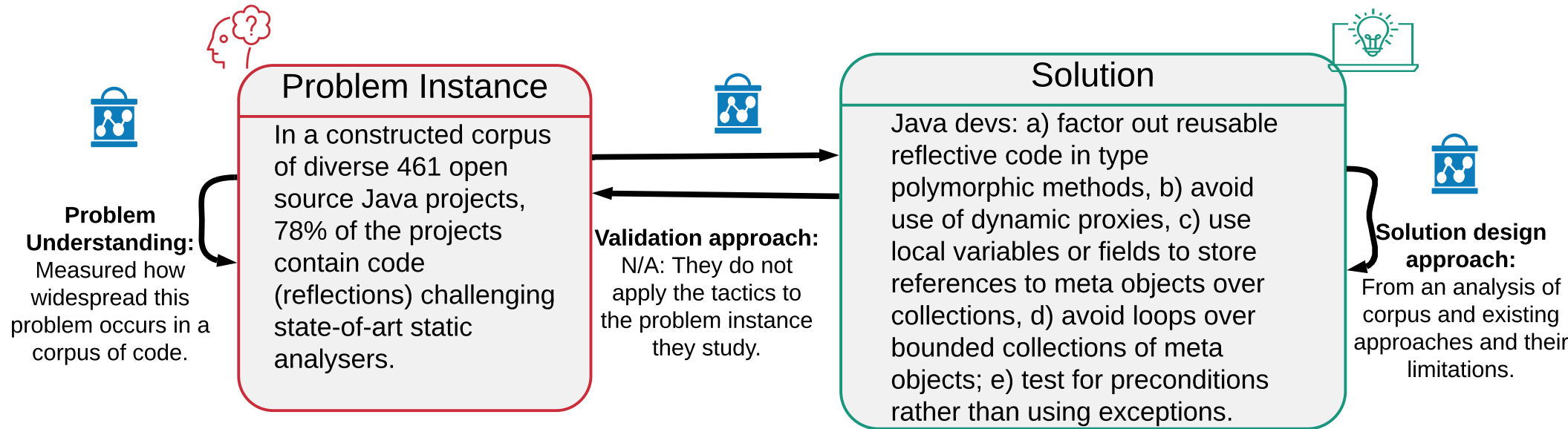
Rigor: Constant comparisons and theoretical coding were used as part of the grounded theory approach.



Novelty: A theory of 5 dimensions (culture, software practices, team practices, management approach & reflective practices) to model how a team is transitioning to agile



Technological rule: To improve or enable static analysis of Java projects, when reflective code is used, follow the guidelines provided in this paper.



Relevance: For Java software engineers prioritizing on robustness and for static analysis tool builders wishing to improve their tools.



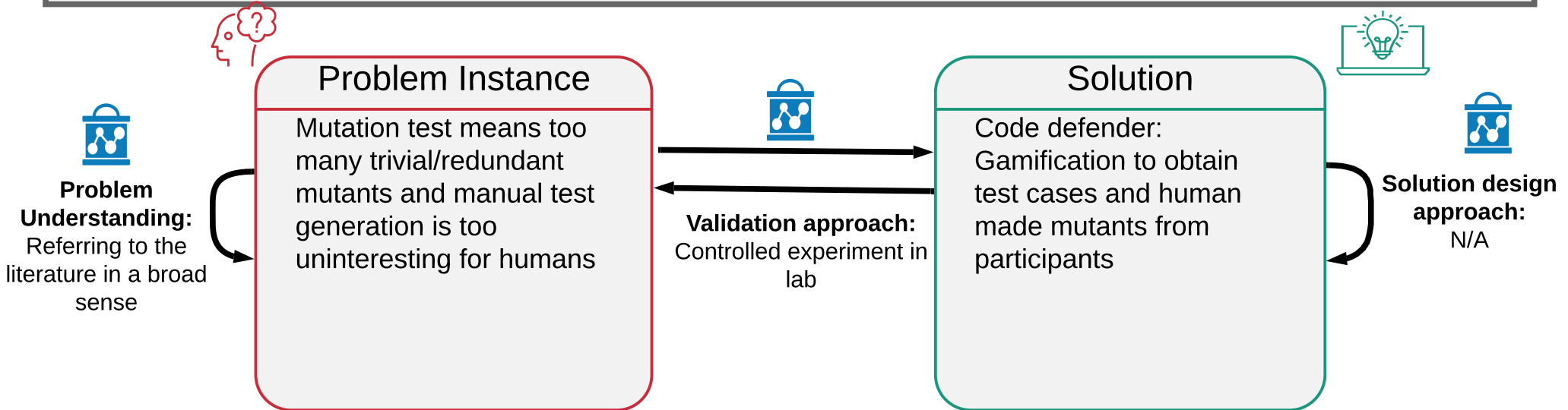
Rigor: A literature review of existing approaches for static analysis and a survey of real world projects' use of the Reflection API



Novelty: New strategies for dealing with analysis/authoring of reflective code



Technological rule: To increase efficient replication and aggregation of knowledge in software engineering research, developing tools, change academic publication and rewarding structures



Relevance: A very general class of issues with respect to mutation testing and automated test generation



Rigor: The evaluation is conducted in an artificially created instance, where rigorous experimental procedures are applied.



Novelty: Code Defender, the idea of using gamification in order to obtain better results in carrying out the otherwise rather dull tasks of formulation mutants and test cases.



Technological rule: To reduce wasteful text executions, when tests are poorly placed in modules, use historical build information & test dependencies to guide which tests should be run.



Problem Instance

Many large software projects have imprecise dependency graphs that lead to wasteful test executions which impacts on developer productivity

Problem Understanding

The authors mathematically formalize the problem of wasteful test executions that occur when tests are poorly placed in project modules.



Solution

TestOptimizer, a technique which uses a greedy algorithm to suggest test movements between test nodes providing a ranked list of suggestions to devs allowing them to choose and reduce the expected number of test executions.



Solution design approach:

A greedy algorithm is proposed to reduce # of test executions by suggesting test movements that consider historical build information & test dependencies

Validation approach:
TestOptimizer was applied to 5 large proprietary projects, calculating time saved by potential movements. Validation was carried out with developers who accepted 84% of the suggestions.



Relevance: Relevant for developers during regression testing of large systems when tests may be poorly positioned, leading to wasteful test executions at build time.



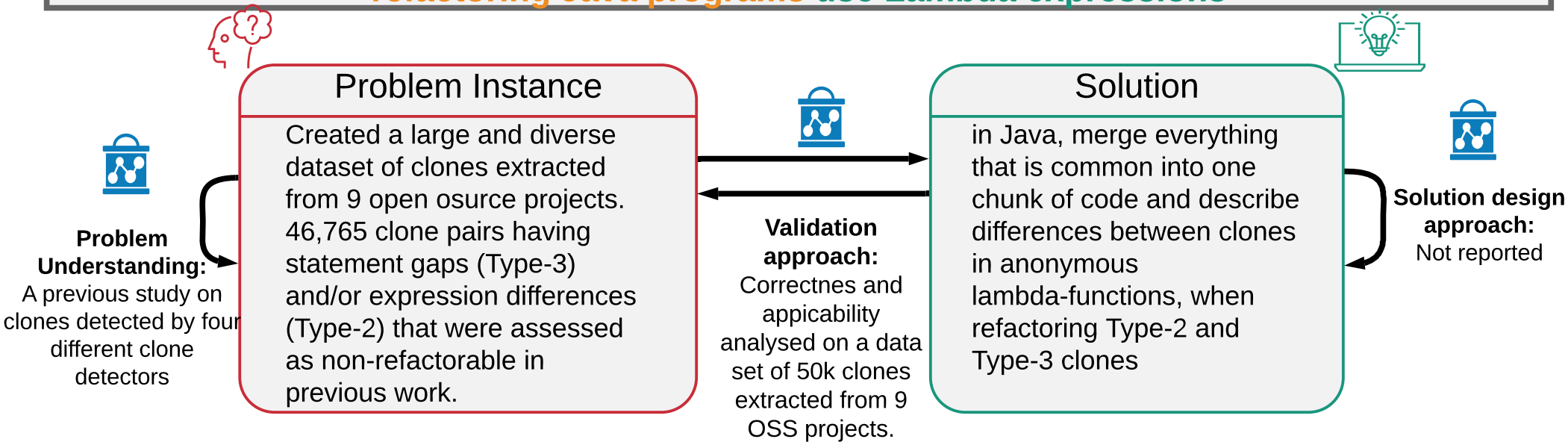
Rigor: Case study with 5 large systems was carried out.



Novelty: Proposal of TestOptimizer, a tool for integration in a build environment for making suggestions on how to reduce wasteful test executions. The tool can also be used to reduce impact of flaky tests.



Technological rule: To enable the refactoring of Type-2 and Type-3 clones having behavioral differences that cannot be parameterized with regular parameters when refactoring Java programs use Lambda expressions



Relevance: Java-program with type 2 and type 3 clones



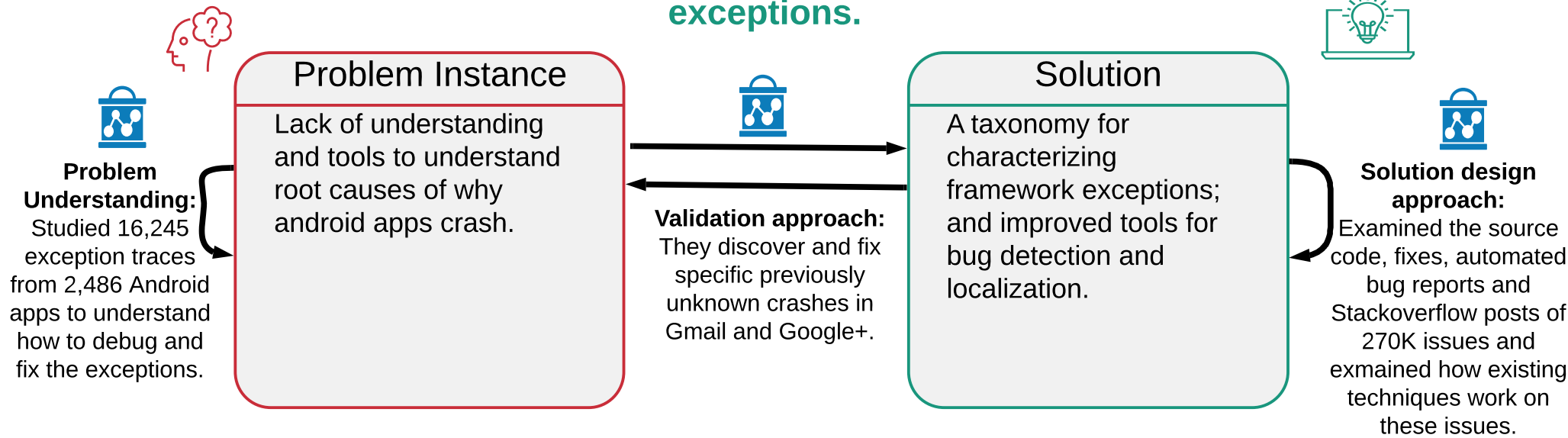
Rigor: By applying it and evaluating its effect



Novelty: A technique and tool that utilizes Lambda expressions to enable the refactoring of Type-2 and Type-3 clones



Technological rule: To identify and fix framework exceptions in a more timely manner in android apps use an enhanced dynamic testing tool to detect bugs and a framework exception localization tool to explain the root cause of framework exceptions.



Relevance: Improve tools for android app developers to handle framework exceptions during development and testing.



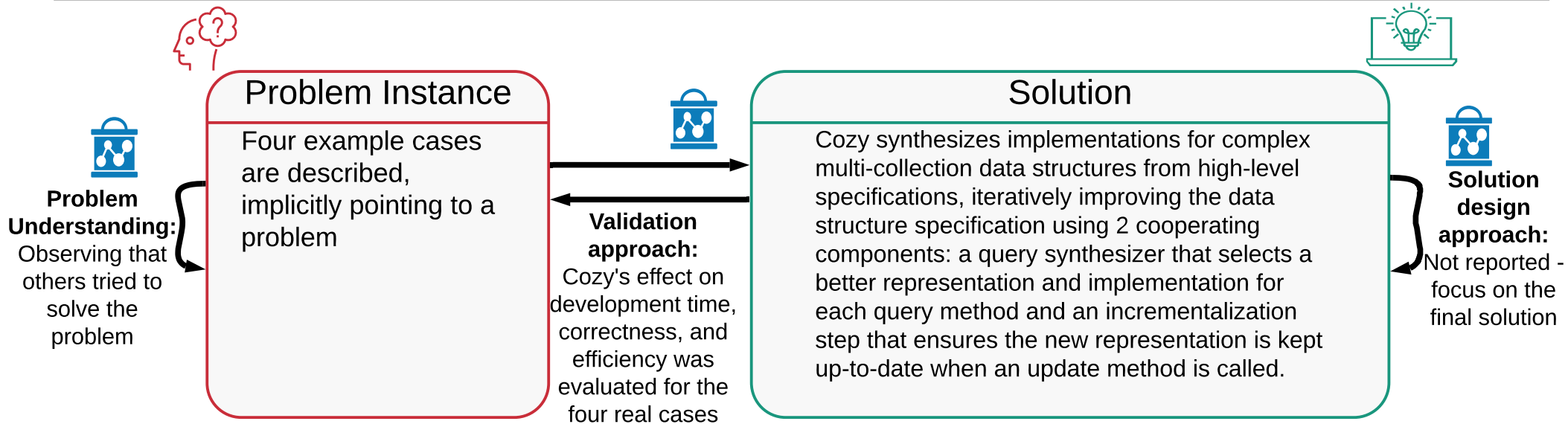
Rigor: Used their tools for revealing and speeding up the fix of of previously unknown bugs in real world apps.



Novelty: Show extent of framework exceptions in Android Apps, contributing a dataset and two prototype tools for revealing and localizing such exceptions.



Technological rule: To synthesize data structures that track subsets and aggregations of multiple related collections alternate steps of query synthesis and incrementalization



Relevance: Data structure problems, especially in domains like user interfaces or web services where software must manage some internal state and also handle asynchronous events.



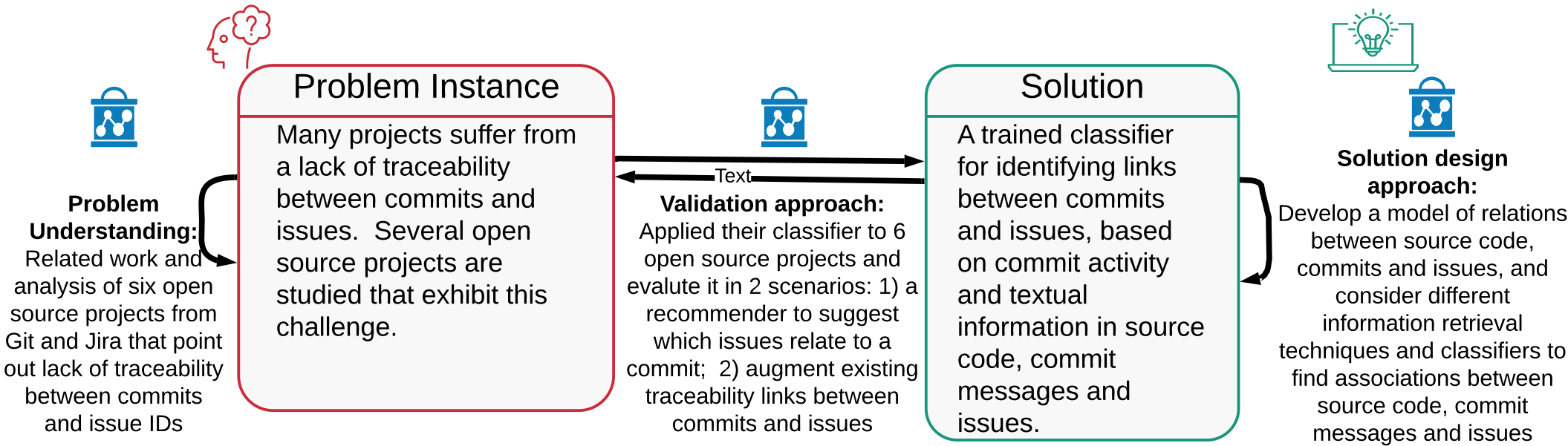
Rigor: Proof of concept demonstrated by applying the solution to four real cases



Novelty: It is a new technique for data structure synthesis that overcomes many of the limitations of previous work



Technological rule: To improve the traceability of commits to issues due to missing links during and after the commit process use the Link Classifier



Relevance: They analyzed six OSS showing that an average of only about 60% of commits were linked to specific issues.



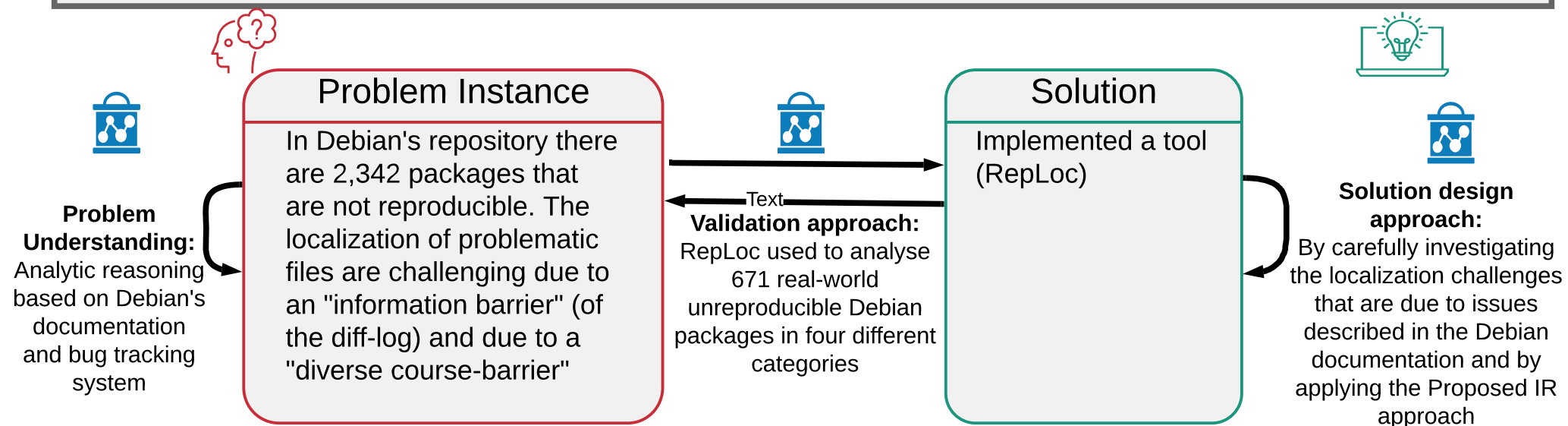
Rigor: The proposed solution is applied to the projects to show that it can find links that already exist as well as recommend non existing links that were manually verified.



Novelty: The approach proposed - Link Classifier - that trains a classifier to recommend links when commits are made and augments an existing set of commits and issues with automatically identified links.



Technological rule: To effectively localize the problematic files for unreproducible builds use the automated framework to analyse differences between binaries built in different build environments



Relevance: Relevant in projects where localisation of problematic files is currently inefficient



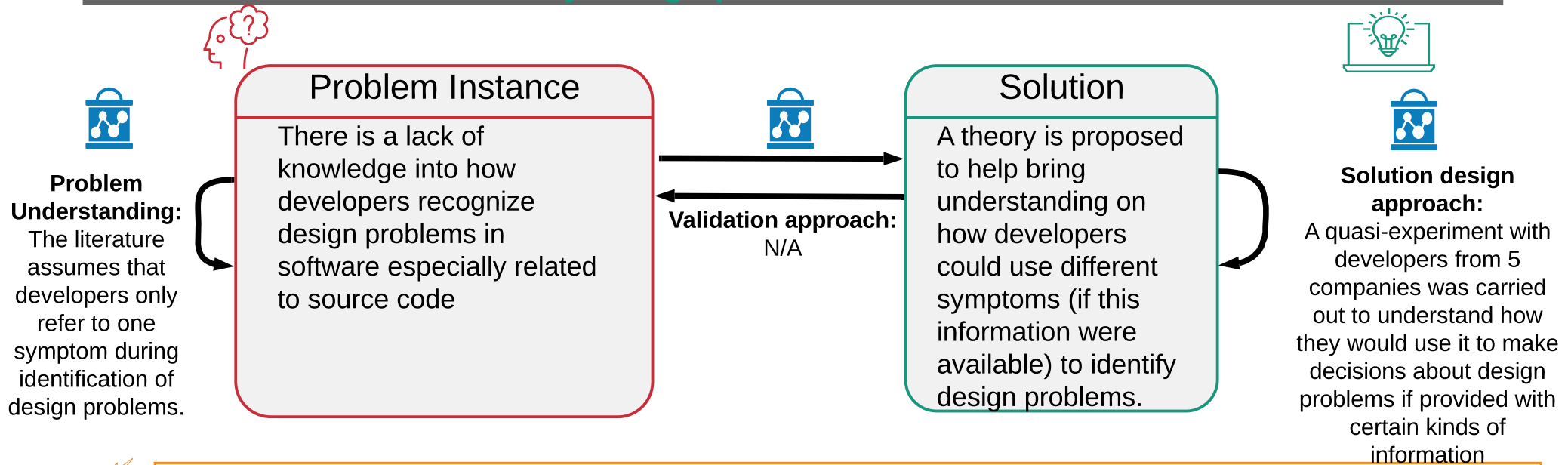
Rigor: Implemented a prototype and conducted extensive experiments over 671 real-world unreproducible Debian packages in 4 different categories. RepLoc achieves accuracy rate of 47% (only accounting for the top most rated files). If they expand the examination to top 10 ranked files in the list, the accuracy rate becomes 79%



Novelty: The first work to address the localization task for unreproducible builds



Technological rule: In order to analyze symptoms that manifest in code elements during source code analysis use the grounded theory that explains how developers identify design problems in source code



Relevance: Field study with developers from 5 companies that may encounter design problems.



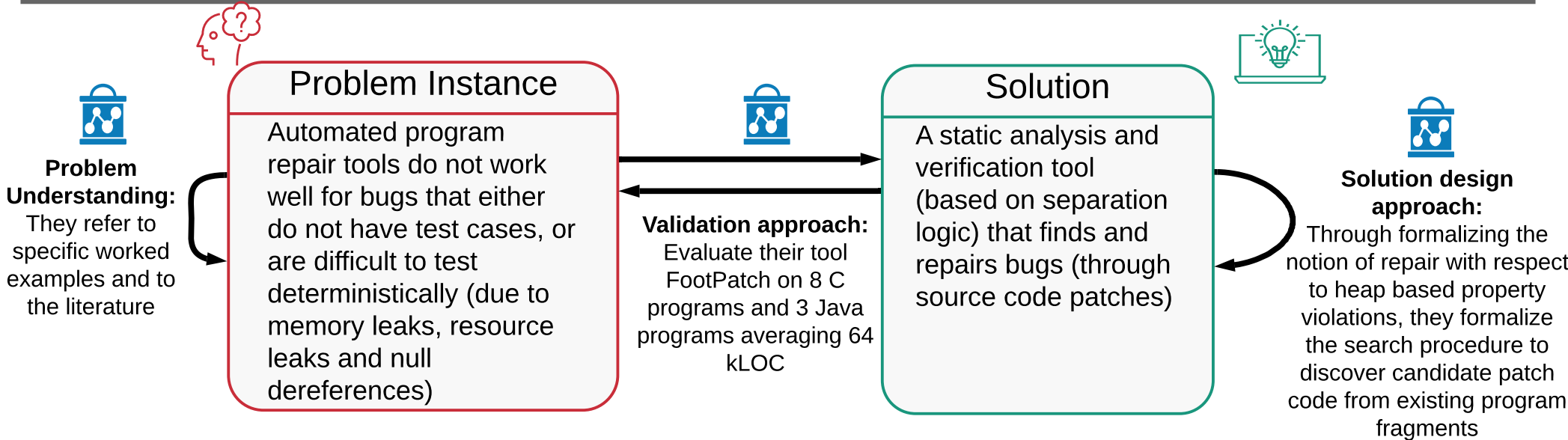
Rigor: Quasi experiment with professionals from 5 software companies to build a grounded theory that explains how developers identify design problems in practice



Novelty: Grounded theory of how developers diagnose design problems



Technological rule: To automatically repair resource leaks, memory leaks and null dereferences in source code that lacks test cases or developer annotations use separation logic to generate patches based on code fragments from the program under repair



Relevance: They apply their tool (FootPatch) by correctly fixing 55 bugs (11 previously unknown) in 11 selected multi-language projects in an efficient manner



Rigor: Evaluation on real software projects



Novelty: A new static automatic program repair technique based on Separation logic that generates patches for certain types of bugs (resource leaks, memory leaks and null dereferences) that does not rely on preexisting test cases or developer annotations



Technological rule: In order to face the problem of testing strategies when conducting concolic testing adopt the Optimal Strategy and use the Greedy Algorithm for identifying the optimal policy with low complexity



Problem Understanding:
Identify existing heuristics from literature

Problem Instance

Concolic execution provides a combination of symbolic execution and concrete testing. However, what is the optimal concolic testing strategy is still an open problem



Validation approach:
Conduct an evaluation of the Greedy Algorithm by using the approach on programs in GNU Scientific Library

Solution

They develop a framework which helps to define and compute the optimal concolic testing strategy based on Markov decision processes



Solution design approach:
Authors conduct experiments to compare the performance of the optimal strategy vs. heuristic-based ones, to investigate whether existing heuristics are reasonably effective



Relevance: Results of experiments point out that all existing heuristics have significantly higher costs than the optimal cost



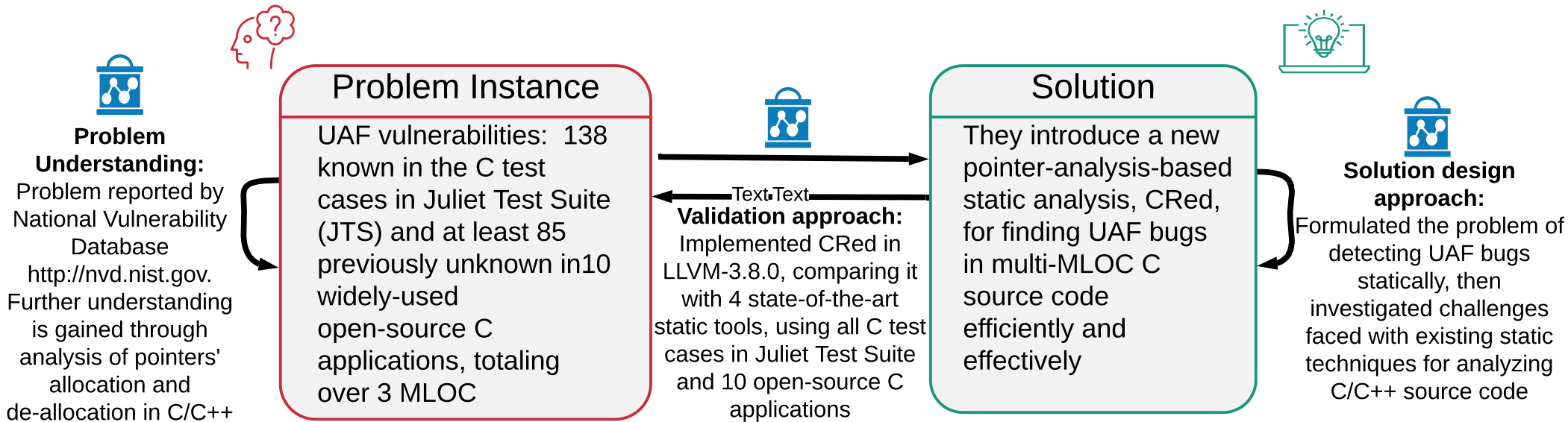
Rigor: Experiments that compare the performance of the optimal strategy vs heuristics based ones



Novelty: Proposal of a framework to derive optimal concolic testing strategies, and analyze existing heuristics; proposal of a Greedy algorithm to approximate the optimal strategy.



Technological rule: To mitigate for Zero-day Use-After-Free (UAF) vulnerabilities In standards compliant C-programs apply pointer-analysis-based static analysis



Relevance: Zero-day Use-After-Free (UAF) vulnerabilities are increasingly popular and highly dangerous, but few mitigations exist



Rigor: Investigation of four alternative solutions was done



Novelty: Three contributions: (i) a spatio-temporal context reduction technique, (ii) a multi-stage analysis for filtering out false alarms efficiently, and (iii) a path-sensitive demand-driven approach for finding the points-to information required